MINISTORY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF TECHINICAL AND VOCATIONAL EDUCATION

GOVERNMENT TECHINICAL INSTITUTE (INSEIN)

DEPARTMENT OF ELECTRONIC ENGINEERING

PROJECT REPORT PAPER ON
"SMART HOME USING IOT CONTROL"

REPERSENTED BY

2GEC Students

(2022-2023)Academic Year

November, 2023

MINISTRY OF SCIENCE AND TECHNOLOGY (INSEIN)

DEPARTMENT OF ELECTRONIC ENGINEERING

We certify that we have examined, and for Committee for acceptance a project report entitled: "**SMART HOME USING IOT CONTROL"** submitted by **3GEC-students (November, 2023)** to the Department of Electronic Engineering as the requirements for AGTI (Electronic and Communication).

Board of Examiners:

1. Name - U Aung Kyaw San
   Associate Professor and Head of Department
   Department of Electronic Engineering .....
   (Chairman)

2. Name - Daw Theint Theint Htun
   Associate Professor
   Department of Electronic Engineering .....
   (Supervisor)

3. Name - Daw Ei Ei Khin
   Lecturer
   Department of Electronic Engineering .....
   (Co-Supervisor)

ACKNOWLEDGEMENTS

We would like to extend our sincere gratitude to our esteemed Principal, **U Myat Ko**, for unwavering support, guidance, and belief in our capabilities throughout the execution of this project. His leadership and encouragement have been instrumental in driving us towards excellence.

We also express our heartfelt appreciation to School of Engineering, Professor, **Dr. Ei Phyo Wai**, for continuous encouragement and valuable insights that have significantly contributed to the success of our project.

A special note of thanks to the Head of Department, **U Aung Kyaw San**, Associate Professor for invaluable guidance, expertise, and encouragement, which has been pivotal in shaping our project's direction and success.

We extend our deepest gratitude to our supervisor teachers, **Daw Theint Theint Htun,** Associate Professor and **Daw Ei Ei Khin**, Lecturer for unwavering support, mentorship, and constructive feedback throughout this journey. Their dedication and commitment played an integral role in refining our project and guiding us towards excellence.

Last but not least, we would like to express our heartfelt appreciation to each of our dedicated teammate, for their hard work, collaboration, and dedication. Their contributions and teamwork have been essential in achieving our project goals.

Each of these individuals has played a crucial role in our project's success, and we are immensely grateful for their support, guidance, and encouragement.

ABSTRACT

The abstract summarizes a comprehensive smart home system that combines safety features, convenience mechanisms, and energy-efficient solutions through Arduino Uno and ESP32 microcontrollers, leveraging Blynk Cloud for monitoring and Arduino Cloud for IoT control. This innovative setup integrates an automatic fire alarm system to swiftly respond to fire emergencies, prioritizing household safety. Additionally, the water level control function manages water levels, preventing overflow and ensuring convenience. A sophisticated lighting system contributes to energy conservation and user convenience through IoT-based controls and sensors automation. Moreover, real-time monitoring of temperature, humidity, and door control using ESP32 enables remote management and monitoring of home environments. By utilizing Blynk Cloud for monitoring and Arduino Cloud for IoT control, this interconnected system creates a holistic smart home environment focused on safety, convenience, and energy efficiency.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1. Introduction of Smart Home

Step into a world where home isn't just a place; it's a responsive, intuitive companion designed to make life smoother and more enjoyable. Our Smart Home isn't about intricate technicalities; it's about the seamless integration of innovation into daily routine.

At the core of this transformation is a thoughtful combination of advanced microcontrollers. These unseen heroes power the magic behind the scenes, orchestrating a symphony of automation without requiring you to be a tech expert. Your home becomes an extension of yourself, anticipating your needs and responding with a touch of sophistication.

Through a user-friendly interface, you're in control, effortlessly managing your environment. This isn't just about convenience; it's a pledge to create a living space that mirrors your preferences, prioritizing simplicity and enhancing the moments that matter.

Embark on this journey into the future of living, welcome to a home that doesn't just shelter but understands — a Smart Home that makes life a little brighter, a touch more comfortable, and a lot more enjoyable.

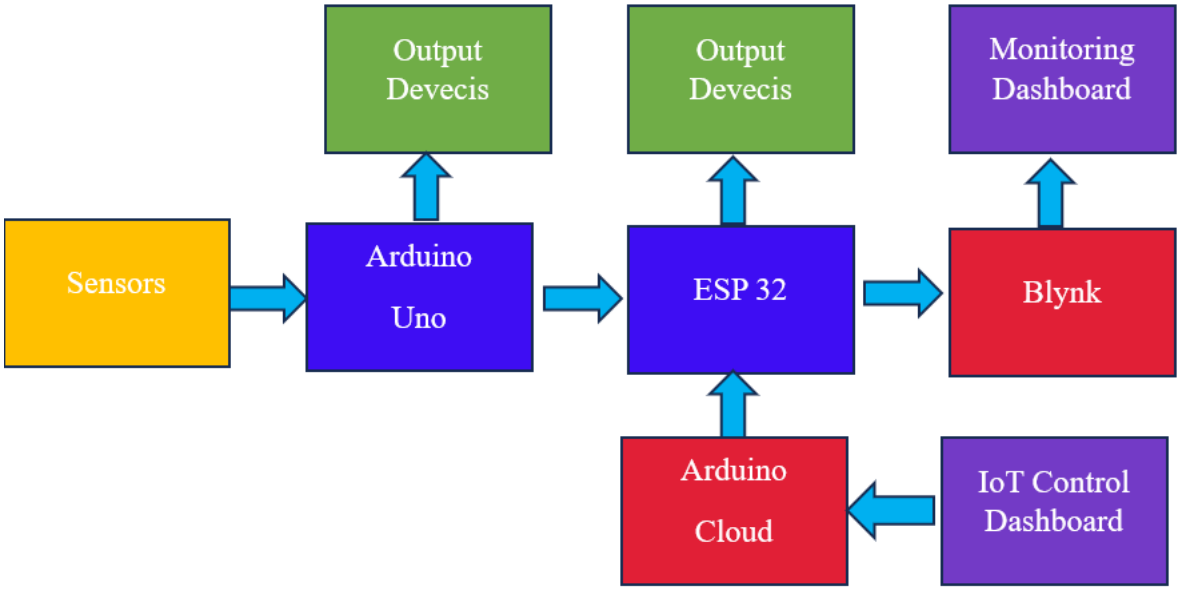


Figure 1.1. Block Diagram of Smart Home using IoT Control

1.2. Aim and Objectives

The whole aim of our smart home is to ensure the safety of home, make lives easier and convenient, save energy, enhance security and control things anytime, anywhere.

- To ensure the safety of home we installed an automatic fire alarm system.
- To make lives easier and convenient, an automatic water level control system is a gift.

- To save energy, we bring automatic and eco-friendly lighting on/off systems.
- To enhance home security, we established an IoT control door lock system.
- To access room lighting anytime, anywhere, we inserted an IoT control lighting system.

1.3. Outline of Project

Introduction is offered in chapter one. Chapter two indicates the features of sensor implementation included in each sector of smart home. Operation procedure of each sector is described technically in the third chapter. Conclusion is gifted to you in the final chapter.

CHAPTER 2
BACKGROUND THEORY

2.1 Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328P, developed in Italy by Arduino. The board features digital and analog input/output pins, a 16 MHz crystal oscillator, a USB connection, a power jack, and ICSP header. The primary goal of the Arduino Uno is to provide an easy and accessible platform for hobbyists, artists, and designers to create interactive projects.

The background theory of the Arduino Uno revolves around the concept of open-source hardware and software. It was designed to be user-friendly and approachable for those with little to no electronics or programming experience. The board can be programmed using the Arduino Software (IDE), which is based on a simplified version of C++.

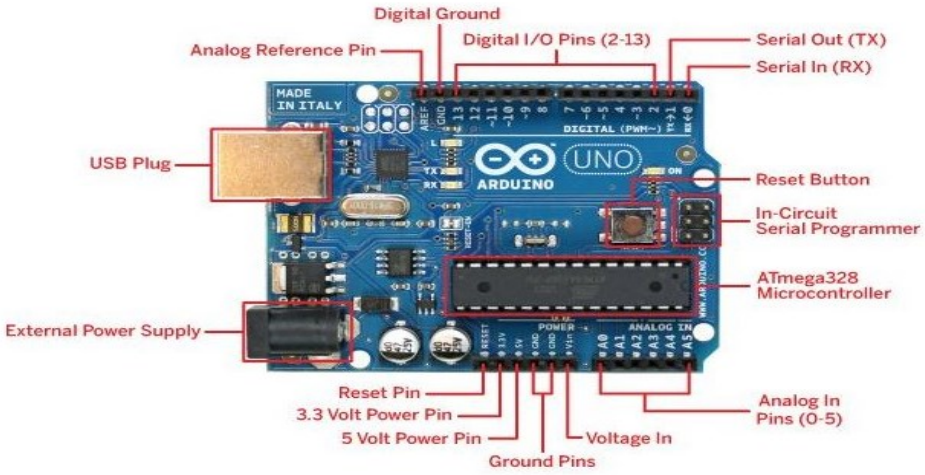


Figure 2.1. Arduino Uno R3 Pin Diagram

The board's microcontroller allows it to interpret inputs from sensors, buttons, and other electronic components and control various outputs such as LEDs, motors, and displays. The Arduino Uno's main processor communicates with these peripherals through its digital and analog input/output pins.

In essence, the background theory of the Arduino Uno involves enabling individuals to easily prototype and create electronic projects without needing extensive knowledge of electronics or programming. It has become widely popular in the maker community and educational settings due to its user-friendly nature and vast community support.

2.2 ESP32

The ESP32 is a powerful microcontroller and system on a chip (SoC) that is designed for Internet of Things (IoT) applications. It is based on the Xtensa LX6 microprocessor, which

features a dual-core CPU operating at up to 240 MHz. The ESP32 also includes integrated Wi-Fi and Bluetooth capabilities, making it suitable for a wide range of connected device applications.

The ESP32 operates on low power and can be used in battery-powered devices. It also features a rich set of peripheral interfaces, including analog-to-digital converters, pulse-width modulation, touch sensors, and more. The ESP32 also supports various communication protocols, such as SPI, I2C, I2S, and UART, making it versatile for interfacing with other devices and sensors.

The background theory of ESP32 involves understanding its architecture, capabilities, and the software development environment. It typically involves programming in the Arduino IDE using the ESP32 core, or using the Espressif IoT Development Framework (ESP-IDF) for more advanced applications. Overall, the ESP32 provides a powerful and flexible platform for developing IoT solutions with wireless connectivity and a wide range of sensor and interface options.

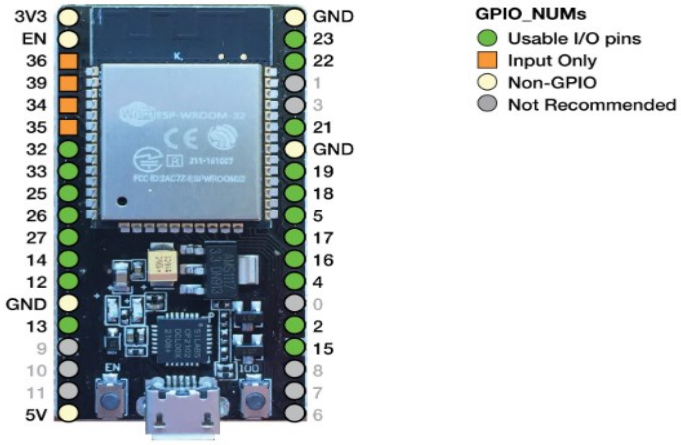


Figure 2.2. ESP32 Pin Diagram

2.2.1 Blynk

Blynk is a platform that offers a range of tools for IoT (Internet of Things) and mobile app development. It allows developers to create custom mobile apps to control hardware, such as microcontrollers and IoT devices, through the use of simple drag-and-drop widgets. The platform is designed to enable rapid prototyping and development of IoT projects, making it accessible to both beginners and experienced developers.

Blynk uses a client-server architecture, where the Blynk server acts as the central hub for communication between the mobile app and the hardware. The mobile app communicates with the Blynk server using the Blynk cloud, which facilitates the exchange of data and commands between the app and the connected devices.

Blynk provides a variety of widgets that can be easily added to the mobile app interface, allowing users to control and monitor connected hardware, such as buttons, sliders, gauges, and graphs. This makes it simple to create custom interfaces for controlling IoT devices and visualizing sensor data.

Overall, the background theory of Blynk revolves around providing an accessible and user-friendly platform for developing IoT applications, with a focus on ease of use and rapid development.

Figure 2.3. Blynk Diagram

2.2.2 Arduino Cloud

The background theory of the Arduino Cloud involves providing a platform for Internet of Things (IoT) projects that allows users to connect their Arduino devices to the cloud. It enables users to remotely manage and monitor their connected devices, create data dashboards, and set up workflows and alerts.

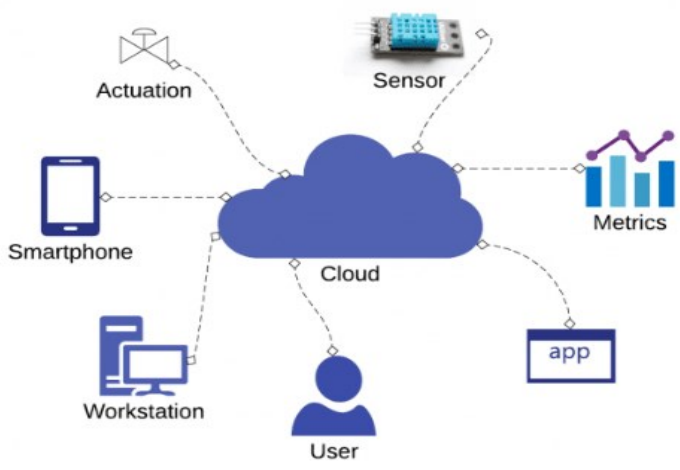


Figure 2.4: Arduino Cloud Diagram

The Arduino Cloud provides a way for Arduino devices, such as the Arduino MKR family of boards, to securely connect to the internet and exchange data with the cloud. It also includes features like over-the-air updates, which allow users to remotely update the firmware of their devices without physically connecting to them.

In terms of technology, the Arduino Cloud uses various protocols and technologies, such as MQTT (Message Queuing Telemetry Transport) for communication between devices and the cloud, secure connections using TLS/SSL, and RESTful APIs for data exchange and device management. Overall, the background theory of the Arduino Cloud revolves around simplifying the process of connecting Arduino devices to the internet and providing a user-friendly platform for managing and monitoring IoT projects.

2.3 Input Devices

Input Devices Lists are-

- Flame Sensor
- MQ2 Smoke Sensor
- Ultrasonic Sensor
- PIR Motion Sensor (Passive Infrared Sensor)
- LDR Sensor Module
- DHT11 Temperature & Humidity Sensor

2.3.1 Flame Sensor

This type of infrared flame sensors is usually incorporated with a standard measurement using LM393 comparator, which allows to obtain the reading in both analog and digital outputs when a certain threshold is exceeded, which is regulated through a potentiometer located in the plate.

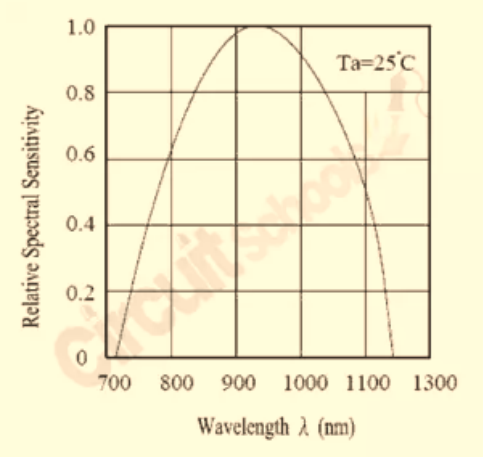


Figure 2.5

As the wavelength of this cheap and small flame sensors are will not perform well when compared to industrial sensors. These may even be affected by indoor lighting which can lead to numerous false positives.

Therefore, the sensitivity and reliability of these cheap flame sensors are not enough to consider them a true security device, although they can be interesting in small electronics projects and for educational purposes, such as sounding an alarm or activating an LED when detecting the flame of a lighter.

Flame sensor Specifications

- Operating Voltage: 3.3V – 5V DC
- Detection range: 60 °
- Detectable wavelength: 760 – 1100 nm
- Working temperature: -25 ° C to 85 ° C
- LM393 in onboard comparator mode
- 1 digital output
- 1 analog output
- Potentiometer to adjust sensitivity

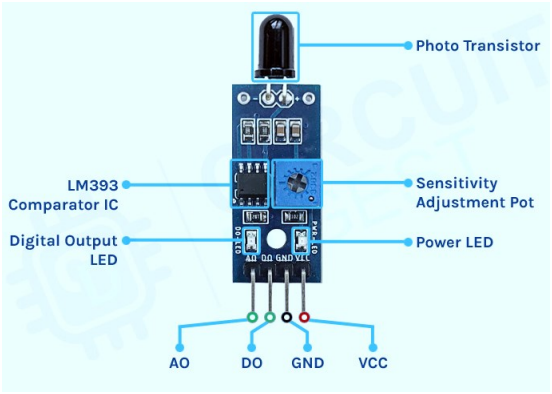


Figure 2.6. Flame Sensor Diagram

2.3.2 MQ2 Smoke Sensor

This Analog Smoke/LPG/CO Gas Sensor module utilizes MQ-2 as the gas detecting component and has a protection resistor and an adjustable resistor on board. MQ2 Gas Sensor module is useful for gas leakage detecting in home and industry. It can detect LPG, i-butane, methane, alcohol, Hydrogen and smoke. Based on its fast response time, measurements can be taken as soon as possible. Also, the sensitivity can be adjusted by the potentiometer.

The voltage that the sensor output changes according to the smoke/gas level that exists in the atmosphere. The sensor outputs a voltage that is proportional to the concentration of smoke/gas.

In other words, the relationship between voltage and gas concentration is the following:

- The greater the gas concentration, the greater the output voltage
- The lower the gas concentration, the lower the output voltage

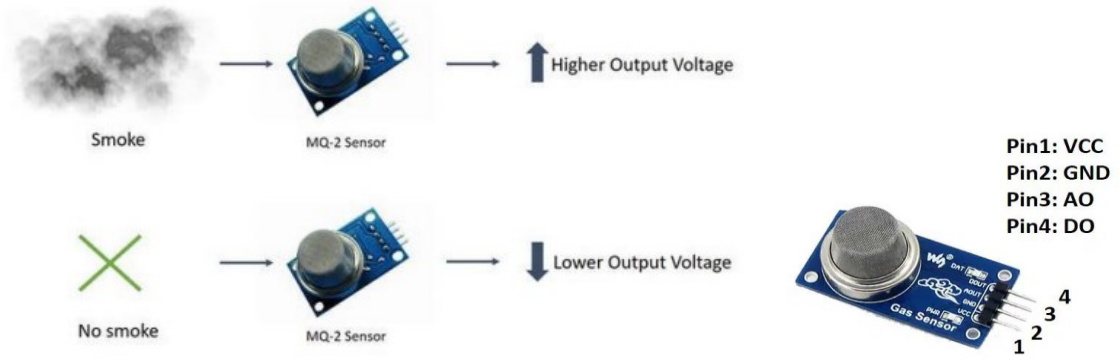


Figure 2.7. MQ2 Smoke Sensor

The output can be an analog signal (A0) that can be read with an analog input of the Arduino or a digital output (D0) that can be read with a digital input of the Arduino.

MQ2 Gas Sensor Specifications

- Power Supply: 5VDC.
- Working Current: ~150mA
- Detecting: LPG, i-butane, propane, methane, alcohol, Hydrogen, smoke
- Warm-up time: ~20-seconds for stable output.
- TTL Digital Ouput: 0V (Low) & 5V (High).
- Analog Output A0: 0.1 ~ 0.3 V.
- Connector: 4-pins header with 2.54mm pitch.
- Dimension: (32 x 20 x 22) mm.
- Weight 8.5g.
- Concentration: 300-10000ppm

2.3.3 Ultrasonic Sensor

Ultrasonic sensors work by sending out a sound wave at a frequency above the range of human hearing. The transducer of the sensor acts as a microphone to receive and send the ultrasonic sound. Our ultrasonic sensors, like many others, use a single transducer to send a pulse and to receive the echo. The sensor determines the distance to a target by measuring time lapses between the sending and receiving of the ultrasonic pulse.

The working principle of this module is simple. It sends an ultrasonic pulse out at 40 kHz, which travels through the air, and if there is an obstacle or object, it will bounce back to

the sensor. By calculating the travel time and the speed of sound, the distance can be calculated.

Formula:

Distance = Speed * Time

Distance = Speed of Sound in Air * (Time Taken / 2)

Speed of sound in air = 344 m/s.

Ultrasonic sensors are a great solution for the detection of clear objects. For liquid level measurement, applications that use infrared sensors, for instance, struggle with this particular use case because of target translucence.

For presence detection, ultrasonic sensors detect objects regardless of color, surface, or material (unless the material is very soft, like wool, as it would absorb sound).

To detect transparent and other items where optical technologies may fail, ultrasonic sensors are a reliable choice.

Ultrasonic Sensor Specifications

- Operating Voltage: +5V
- Operating Frequency: 40Hz
- Operating Current: <15mA
- Minimum Detection Range: 2cm
- Maximum Detection Range: 400cm
- The sensor contains 4 pins: GND, VCC, Echo, Trigger
- The Trigger and Echo pins act as an input and output pins respectively
- Trigger pin is used to trigger ultrasonic burst out of the sensor.
- Echo pin will send out a pulse whose width is equal to the time taken for the ultrasonic burst to come back after reflecting from an obstacle.

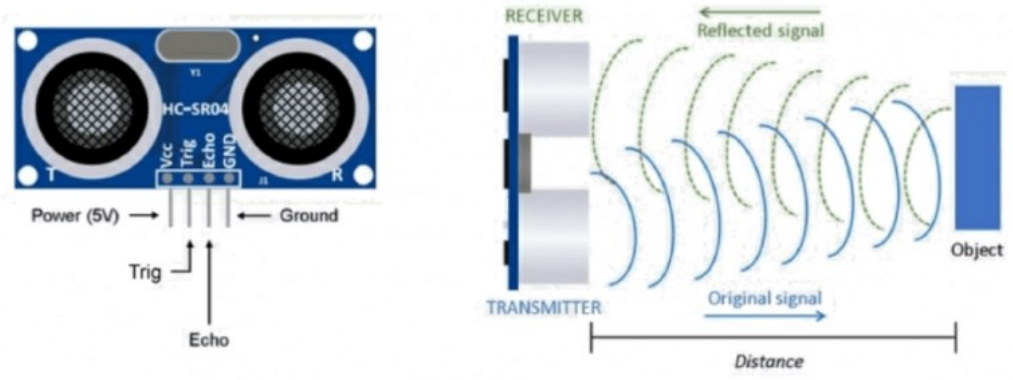


Figure 2.8: Ultrasonic Sensor Diagram

2.3.4 PIR Motion Sensor (Passive Infrared Sensor)

PIR sensor definition is; a sensor that is used to measure infrared light radiating from objects like the human body or animals. In PIR sensor, the term PIR stands for “passive infrared sensor”. PIR sensors can detect the movement of an animal or human within a fixed range. Generally, all objects at above zero temperatures produce heat energy in the IR radiation form. So if the object is hotter then it emits more radiation. So this kind of radiation is not observable to the human eye as it is emitted at IR wavelengths. So PIR sensor is particularly designed for detecting such infrared radiation levels. These sensors are most frequently utilized in motion detectors, security alarms & automatic lighting-based applications.

A PIR sensor includes two main parts like pyroelectric sensor and fresnel lens. In the following diagram, the sensor is a round metal including a rectangular crystal within the center. A fresnel lens is a special lens that focuses the IR signals on the pyroelectric sensor. Here, the pyroelectric sensor is capable of detecting different infrared radiation levels.

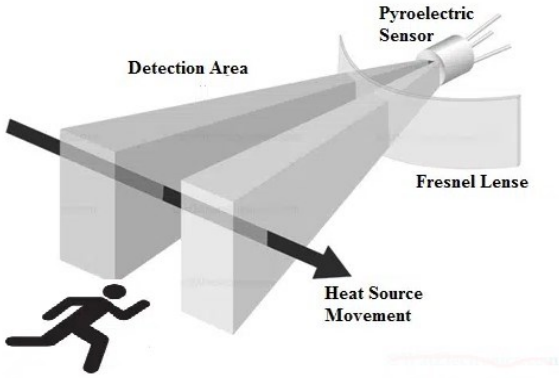


Figure 2.9. PIR Sensor Working

PIR Motion Sensor Specifications

The specifications of the PIR sensor include the following.

- The recommended input voltage supply is +5V.
- The output voltage is 3.3V.
- It can differentiate between the movement of an object & human.
- Operating modes are repeatable & non-repeatable.
- The current drain is <60uA.
- The detection angle is <140°
- The detection distance is 3 to 7m.
- Blockade time by default is 2.5s.
- Working temperature ranges from -20-+80°C.
- Low power utilization – 65mA.

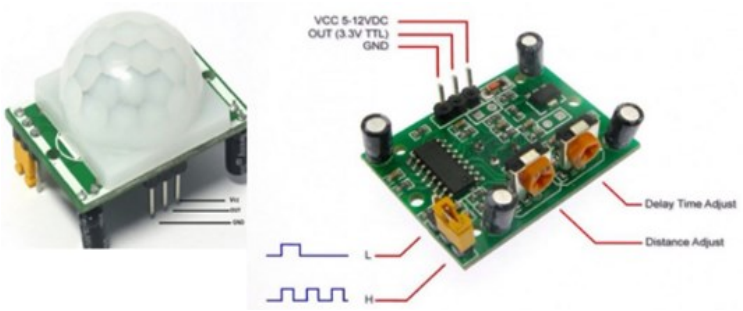


Figure 2.10. PIR Motion Sensor (Passive Infrared Sensor) Diagram

2.3.5 LDR Sensor Module

First of all, we need to connect the LDR sensor module to a 5v power supply. Then set the threshold voltage at the non-inverting input (3) of the IC according to the present light intensity by rotating the preset knob for setting the sensor sensitivity.

When light intensity increases on the surface of the LDR then the resistance of the LDR decreases. Then the maximum amount of voltage will be allocated across the resistor(R3). So, a Low amount of voltage from the LDR is given to the Inverting input (2) of the IC. Then the Comparator IC compares this voltage with the threshold voltage. In this condition, this input voltage is less than the threshold voltage, so the sensor output goes LOW (0).

In contrast, when light intensity decreases (low/dark) on the surface of the LDR then the resistance of the LDR increases. Then the maximum amount of voltage will be allocated across the LDR (R2). So, a high amount of voltage from the LDR is given to the Inverting input (2) of the IC. Then the Comparator IC compares this voltage with the threshold voltage. In this condition, this input voltage is greater than the threshold voltage, so the sensor output goes High (1).

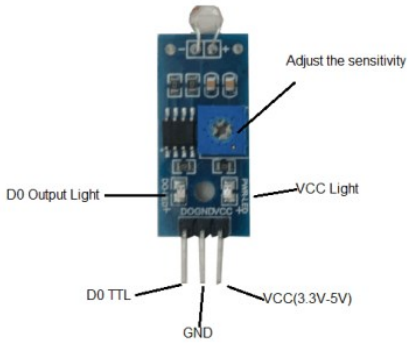


Figure 2.11. LDR Sensor Module

LDR Sensor Module Specifications

- Operating voltage: 5V or 3.3V DC
- Comparator chip: LM393
- Module Pins: 3 pins
- Output type: Digital outputs (D0)

- Sensitivity: Adjustable
- Indicator LED : Output and power LED indicator
- PCB size: 3cm * 1.6cm
- Fixed Hole Diameter: 3mm

2.3.6 DHT11 Temperature & Humidity Sensor

DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measure, process this changed resistance values and change them into digital form.

For measuring temperature this sensor uses a Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with increase in temperature. To get larger resistance value even for the smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

The temperature range of DHT11 is from 0 to 50 degree Celsius with a 2-degree accuracy. Humidity range of this sensor is from 20 to 80% with 5% accuracy. The sampling rate of this sensor is 1Hz.i.e., it gives one reading for every second. DHT11 is small in size with operating voltage from 3 to 5 volts. The maximum current used while measuring is 2.5mA.

DHT11 Temperature & Humidity Sensor Specifications

- Operating Voltage: 3.5V to 5.5V
- Operating current: 0.3mA (measuring) 60uA (standby)
- Output: Serial data
- Temperature Range: 0°C to 50°C
- Humidity Range: 20% to 90%
- Resolution: Temperature and Humidity both are 16-bit
- Accuracy: ±1°C and ±1%

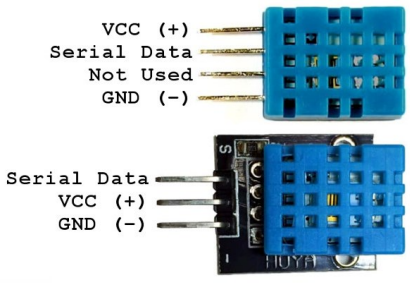


Figure 2.12. DHT11 Temperature & Humidity Sensor Diagram

2.4 Output Devices

Output Devices Lists are -

- Buzzer Module
- L293D Motor Driver Module
- 5V DC Pump Motor
- LEDs

2.4.1 Buzzer Module

The working principle of a buzzer depends on the theory that, once the voltage is given across a piezoelectric material, then a pressure difference is produced. A piezo type includes piezo crystals among two conductors.

Once a potential disparity is given across these crystals, then they thrust one conductor & drag the additional conductor through their internal property. So this continuous action will produce a sharp sound signal.

Buzzer Module Specifications

- Color is black
- The frequency range is 3,300Hz
- Operating Temperature ranges from – 20° C to +60°C
- Operating voltage ranges from 3V to 24V DC
- The sound pressure level is 85dBA or 10cm
- The supply current is below 15mA

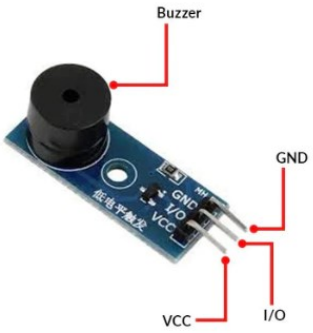


Figure 2.13. Buzzer Module Diagram

2.4.2 L293D Motor Driver Module

A motor driver is basically a current amplifier which takes a low-current signal from the microcontroller and gives out a proportionally higher current signal which can control and drive a motor. In most cases, a transistor can act as a switch and perform this task which drives the motor in a single direction.

Turning a motor ON and OFF requires only one switch to control a single motor in a single direction. What if you want your motor to reverse its direction? The simple answer is to reverse its polarity. This can be achieved by using four switches that are arranged in an intelligent manner such that the circuit not only drives the motor but also controls its direction. Out of many, one of the most common and clever designs is an H-bridge circuit where transistors are arranged in a shape that resembles the English alphabet “H”.

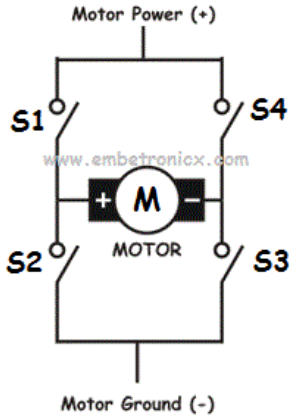


Figure 2.14

A H bridge is an electronic circuit that allows a voltage to be applied across a load in any direction. H-bridge circuits are frequently used in robotics and many other applications to allow DC motors to run forward & backward. These motor control circuits are mostly used in different converters like DC-DC, DC-AC, AC-AC converters, and many other types of power electronic converters. In specific, a bipolar stepper motor is always driven by a motor controller having two H-bridges.

A H-bridge is fabricated with four switches like S1, S2, S3 and S4. When the S1 and S4 switches are closed, then a +ve voltage will be applied across the motor. By opening the switches S1 and S4 and closing the switches S2 and S3, this voltage is inverted, allowing invert operation of the motor.

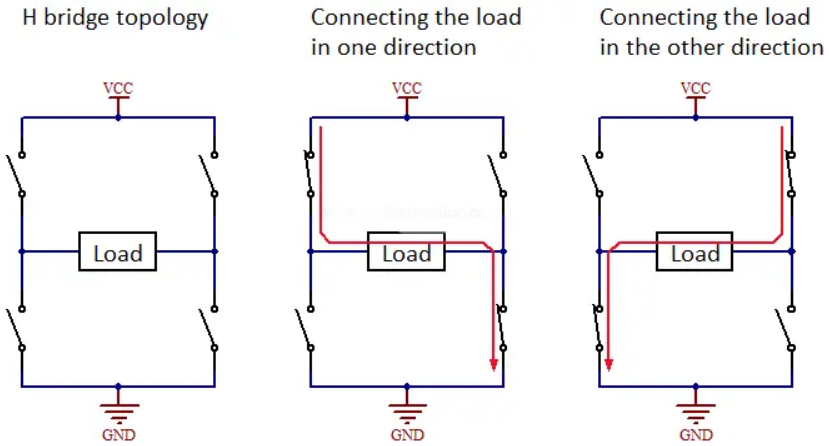


Figure 2.15

Generally, the H-bridge motor driver circuit is used to reverse the direction of the motor and also to brake the motor. When the motor comes to a sudden stop, as the terminals of the motor's are shorted. Or let the motor run free to a stop when the motor is detached from the circuit. The table below gives the different operations with the four switches corresponding to the above circuit.

Table 2.1: L293D Motor Driver Module (H-bridge motor driver)

S1	S2	S3	S4	Operation
1	0	0	1	Motor moves right
0	1	1	0	Motor moves left
0	0	0	0	Motor free runs
0	1	0	1	Motor brakes
1	0	1	0	Motor brakes
1	1	0	0	Short Power Supply
0	0	1	1	Short Power Supply
1	1	1	1	Short Power Supply

L293D Motor Driver Module Specifications

- Can be used to run Two DC motors with the same IC.
- Speed and Direction control is possible
- Motor voltage Vcc2 (Vs): 4.5V to 36V
- Maximum Peak motor current: 1.2A
- Maximum Continuous Motor Current: 600mA
- Supply Voltage to Vcc1(vss): 4.5V to 7V
- Transition time: 300ns (at 5Vand 24V)
- Automatic Thermal shutdown is available
- Available in 16-pin DIP, TSSOP, SOIC packages.

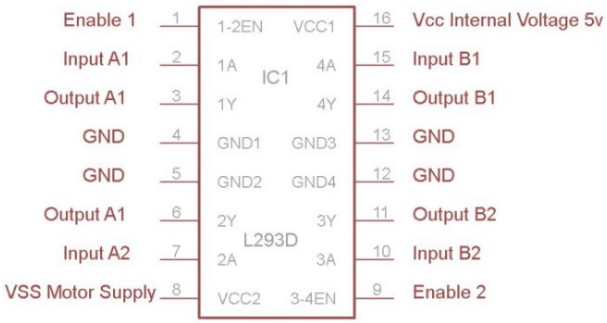


Figure 2.16. L293D Pin Configuration

2.4.3 5V DC Pump Motor

This is a low cost, small size Submersible Pump Motor which can be operated from a 5V power supply. It can take up to 120 liters per hour with very low current consumption of 220mA. Just connect tube pipe to the motor outlet, submerge it in water and power it. Make sure that the water level is always higher than the motor. Dry run may damage the motor due to heating and it will also produce noise.

5V DC Pump Motor Specifications

- Operating Voltage: 5V.
- Operating Current: 0.6A.
- Water Inlet Pipe Diameter: 17mm.
- Water Outlet Pipe Diameter: 8mm.
- Length: 90mm.

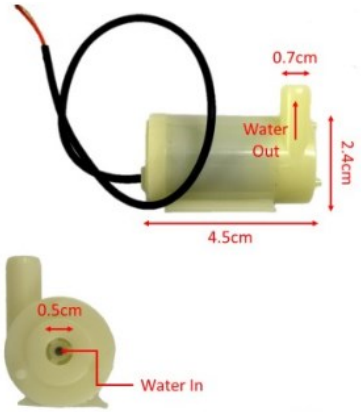


Figure 2.17. 5V DC Pump Motor

2.4.4 LEDs

Like an ordinary diode, the LED diode works when it is forward biased. In this case, the n-type semiconductor is heavily doped than the p-type forming the p-n junction. When it is forward biased, the potential barrier gets reduced and the electrons and holes combine at the depletion layer (or active layer), light or photons are emitted or radiated in all directions. A typical figure blow showing light emission due electron-hole pair combining on forward biasing.

The explanation behind the emission of photons in an LED diode lies in the energy band theory of solids. According to this theory, whether the electron-hole combining will give out photons or not depends on whether the material has a direct band gap or indirect band gap. Those semiconductor materials which have a direct band gap are the ones that emit photons. In a direct bandgap material, the bottom of the energy level of conduction band lies directly above the topmost energy level of the valence band on the Energy vs Momentum (wave vector ‘k’) diagram. When electrons and hole recombine, energy E = hν corresponding to the energy

gap △ (eV) is escaped in the form of light energy or photons where h is the Planck's constant and ν is the frequency of light.

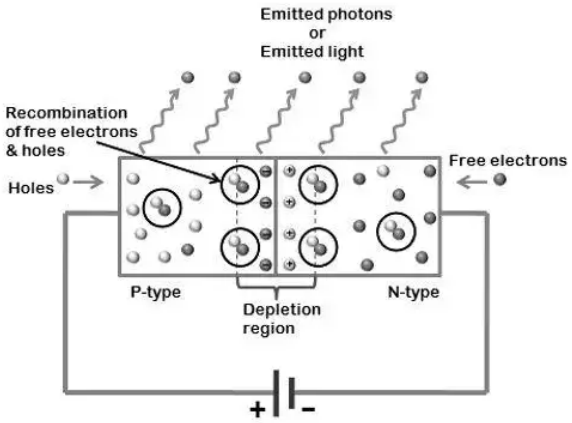


Figure 2.18

Direct Band Gap

While the indirect band gap is non-radiative in nature as the bottom of the conduction band does not coincide with the top of the valence band and the energy corresponding to the energy gap is mostly given in the form of heat. Examples are Si, Ge etc.

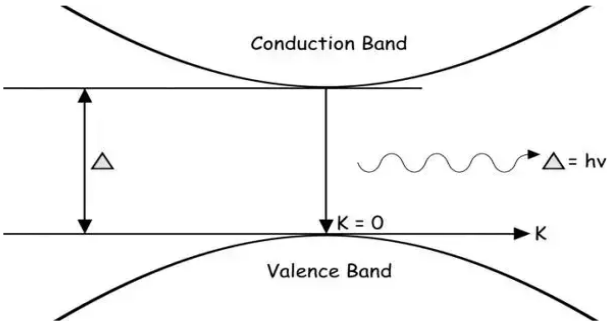


Figure 2.19. Direct Band Gap

Indirect Band Gap

Example of material which has direct band gap is Gallium Arsenide(GaAs), a compound semiconductor which is the material used in LEDs. Dopant atoms are added to GaAs to give out a wide range of colors. Some of the materials used in LEDs are:

- Aluminium Gallium Arsenide(AlGaAs) – infrared.
- Gallium Arsenic Phosphide(GaAsP) – red, orange, yellow.
- Aluminium Gallium Phosphide(AlGaP) – green.
- Indium gallium nitride (InGaN) – blue, blue-green, near UV.
- Zinc Selenide(ZnSe) – blue.

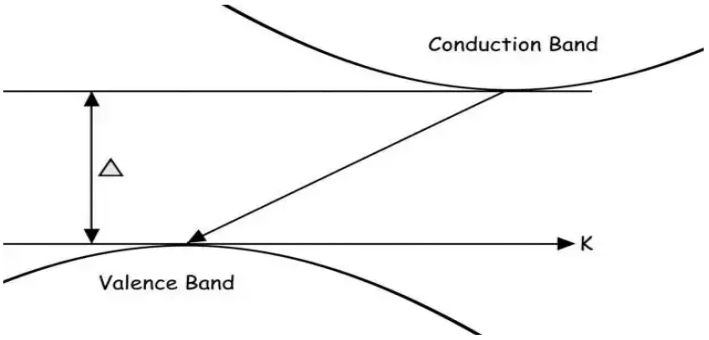


Figure 2.20. Indirect Band Gap

LED Specifications

- Long Life: LEDs can last over 100,000 hours (10+ years) if used at rated specifications.
- No annoying flicker like from fluorescent lamps.
- LEDs are impervious to heat, cold, shock and vibration.
- LEDs do not contain breakable glass.
- Solid-State, shock and vibration resistant.
- Extremely fast turn On/Off times.

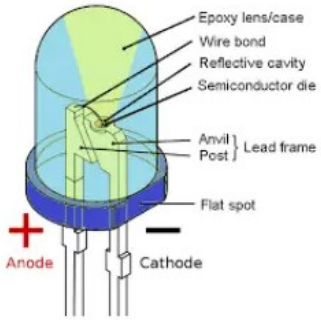
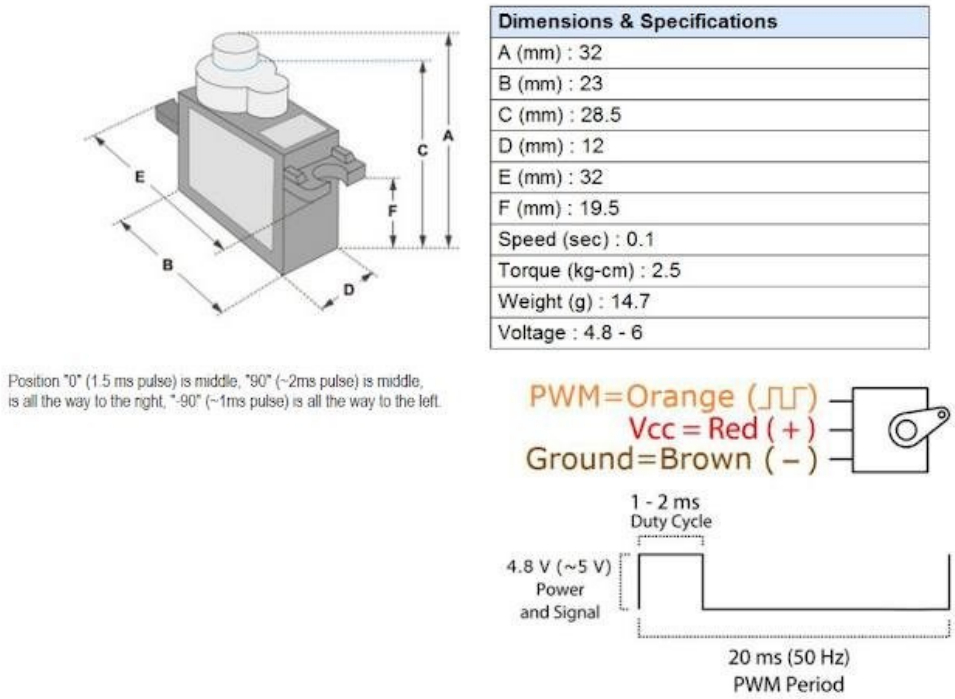


Figure 2.21. LEDs Diagram

2.4.5 Servo Motor

A servo consists of a Motor (DC or AC), a potentiometer, gear assembly, and a controlling circuit. First of all, we use gear assembly to reduce RPM and to increase torque of the motor. Say at initial position of servo motor shaft, the position of the potentiometer knob is such that there is no electrical signal generated at the output port of the potentiometer. Now an electrical signal is given to another input terminal of the error detector amplifier. Now the difference between these two signals, one comes from the potentiometer and another comes from other sources, will be processed in a feedback mechanism and output will be provided in terms of error signal. This error signal acts as the input for motor and motor starts rotating. Now motor shaft is connected with the potentiometer and as the motor rotates so the potentiometer and it will generate a signal. So as the potentiometer's angular position changes,

its output feedback signal changes. After sometime the position of potentiometer reaches at a position that the output of potentiometer is same as external signal provided. At this condition, there will be no output signal from the amplifier to the motor input as there is no difference between external applied signal and the signal generated at potentiometer, and in this situation motor stops rotating.



Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Figure 2.22. Servo Motor Working Principle

Servo Motor Specifications

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 µs
- Temperature range: 0 ºC – 55 ºC
- Position: A 1ms pulse width will correspond to 0°, A 1.5ms pulse width will correspond to 90°, A 2ms pulse width will correspond to 180°.
- correspond to 90°, A 2ms pulse width will correspond to 180°.

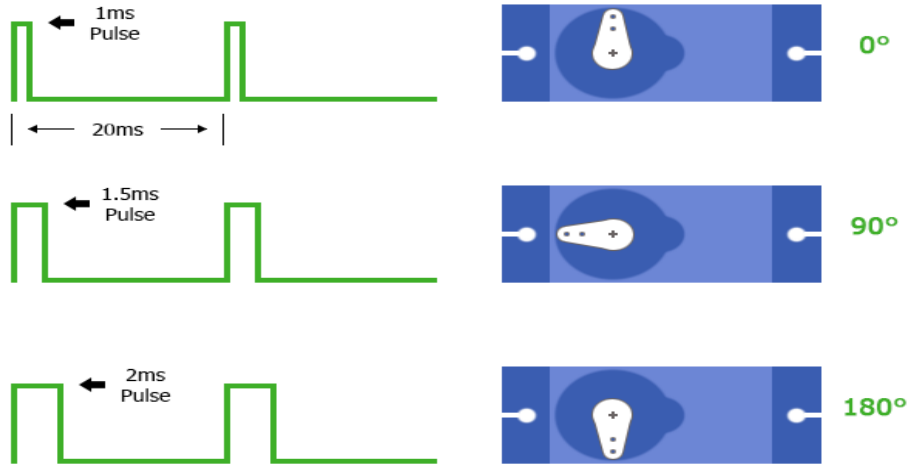


Figure 2.23. pulse width and direction

Figure 2.24. Servo Motor Pin Diagram

CHAPTER 3

OPERATION OF SMART HOME

3.1. Operation procedure of Automatic Fire Alarm System

The operation procedure of an automatic fire alarm system utilizing an Arduino Uno, smoke sensor, flame sensor, and buzzer module involves the following steps.

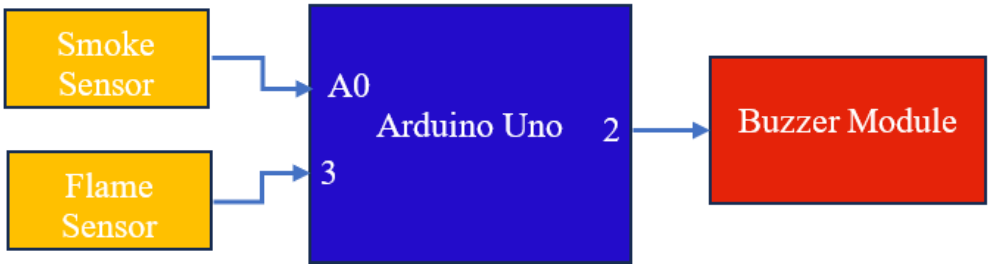


Figure 3.1 Block Diagram of Automatic Fire Alarm System

The smoke sensor and flame sensor continuously monitor their respective environments for any signs of smoke or flames. The Arduino Uno, acting as the central control unit, constantly receives input from the smoke and flame sensors. The Arduino checks the sensor readings. If both the smoke and flame sensors detect high levels simultaneously, indicating a potential fire, the system proceeds to the next step. The Arduino activates the buzzer module to produce a loud alarm sound. The buzzer alerts occupants or nearby individuals to the potential fire hazard. The primary objective of this system is to promptly detect the presence of both smoke and flames, triggering an immediate and audible alert through the buzzer module. This rapid response aims to notify occupants and enable timely actions to mitigate the fire risk. The integration of the Arduino Uno facilitates the processing of sensor data and the coordination of alarm activation, making the system an effective and responsive automatic fire alarm solution.

3.2. Operation Procedure of Water Level Control System

The operation procedure of a water level control system, incorporating an Arduino Uno, L293D mini motor driver module, 5V mini pump motor, and an ultrasonic sensor for distance measurement, involves the following steps.

The ultrasonic sensor continuously measures the distance between its transceiver and the water surface, providing real-time data on the water level. We calculate the distance by converting the signal sent from the echo pin of ultrasonic sensor to Arduino Uno. We use the equation **(T*0.0343) / 2**.

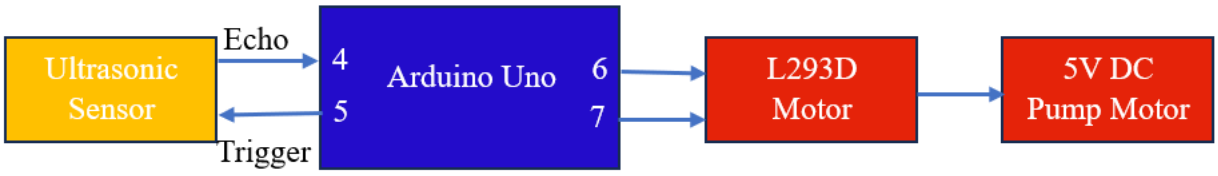


Figure 3.2. Block Diagram of Water Level Control System

The Arduino Uno receives and processes the distance data from the ultrasonic sensor. Based on the distance measurement, the Arduino determines whether the water level is below the lower level, signaling a need to pump water into the container. The water level is below the lower level, the Arduino activates the L293D mini motor driver module. 5V mini pump motor begins pumping water until the ultrasonic sensor indicates that the water level has reached the upper level. The water level is adequate, the Arduino signals the L293D module to stop the motor, ceasing the water pumping process. The primary goal of this system is to maintain a consistent water level by leveraging distance measurements from the ultrasonic sensor. The Arduino Uno and L293D mini motor driver module work in tandem to control the pump motor, ensuring efficient and automated water level management based on real-time measurements.

3.3. Operation Procedure of Automatic Lighting System

The operation procedure of an automatic lighting system involves an LDR (Light Dependent Resistor) module for a compound lamppost and a PIR (Passive Infrared) motion sensor for a bathroom, focusing on energy-saving.

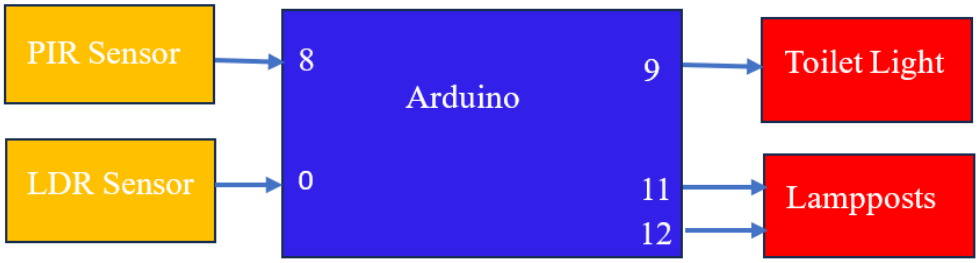


Figure 3.3 Block Diagram of Automatic Lighting System

The PIR motion sensor in the bathroom detects movement, signaling the presence of someone in the area. The LDR module constantly monitors the ambient light level around the compound. The Arduino, serving as the central controller, receives input from both the LDR module and PIR sensor. The system checks for motion detected by the PIR sensor in the bathroom. If motion is detected, indicating someone is in the bathroom, the system turns on the bathroom light. The system checks the ambient light level around the compound using the LDR module. If it falls below a specified threshold, the system proceeds to turn on the compound lamppost. The bathroom light includes timers and the compound lamppost consist of conditions for automatic shutdown. This ensures that the lights turn off after a specified period of inactivity

or when sufficient ambient light is detected. By activating the compound lamppost and bathroom bulb only when necessary (i.e., when it's dark for lampposts, when motion is detected in the bathroom), the system ensures energy-efficient lighting in both areas. This lighting system approach utilizes the LDR module for ambient light sensing around the compound lamppost and the PIR motion sensor in the bathroom for occupancy detection. The goal is to provide automatic, energy-saving lighting in both areas, minimizing unnecessary energy consumption when lights are not needed.

3.4. Operation Procedure of IoT Based Temperature and Humidity Monitoring System

The operation procedure of an IoT-based temperature and humidity monitoring system using ESP32 with the Blynk platform and DHT11 sensor module involves the following steps:

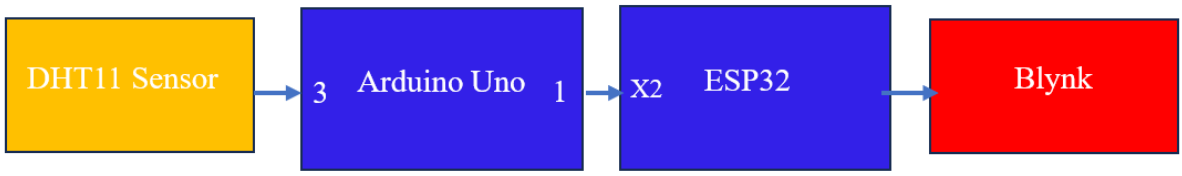


Fig 3.4 Block Diagram of IoT Based Temperature and Humidity Monitoring System

The DHT11 sensor module measures temperature and humidity levels in the environment. The ESP32 microcontroller processes the data from the DHT11 sensor module through the serial communication with Arduino Uno. The ESP32 establishes a connection to the internet using Wi-Fi, enabling communication with the Blynk platform. The ESP32 integrates with the Blynk platform, connecting to the Blynk server through Wi-Fi. The temperature and humidity data collected by the DHT11 sensor are transmitted to the Blynk server in real-time. Users can monitor the temperature and humidity readings in real-time through the Blynk mobile or web application. The primary aim of this system is to provide users with remote and real-time monitoring of temperature and humidity levels. The integration of the ESP32 with the Blynk platform facilitates seamless data transmission and user interaction, making it a comprehensive IoT solution for environmental monitoring.

3.5. Operation Procedure of IoT Administrative Door Control System

The standard operation procedure for IoT administrative door control system using ESP32, Arduino Cloud, and a servo motor is here.

The IoT-controlled door control system operates by utilizing an ESP32 microcontroller, a servo motor, and the Arduino Cloud Remote app. The ESP32 connects to a Wi-Fi network and interfaces with the servo motor to physically lock or unlock the door. The servo motor

manages the door's locking mechanism, while the ESP32 serves as the microcontroller for communication.

Figure 3.4 Block Diagram of IoT Administrative Door Control System

Through the Arduino Cloud platform, the ESP32 establishes a communication link, enabling the Arduino Cloud Remote app to send commands remotely. When a command (e.g., lock or unlock) is initiated from the app, it triggers the ESP32 to activate the servo motor, causing the door lock mechanism to engage or disengage accordingly. This system leverages wireless connectivity, allowing users to control the door lock remotely using the app's interface while the ESP32 interprets and executes these commands, facilitating secure and convenient access to the door.

3.6. Operation Procedure of IoT Control Lighting System

Here is the operation procedure for an IoT-controlled lighting system using ESP32, Arduino Cloud, and LEDs.

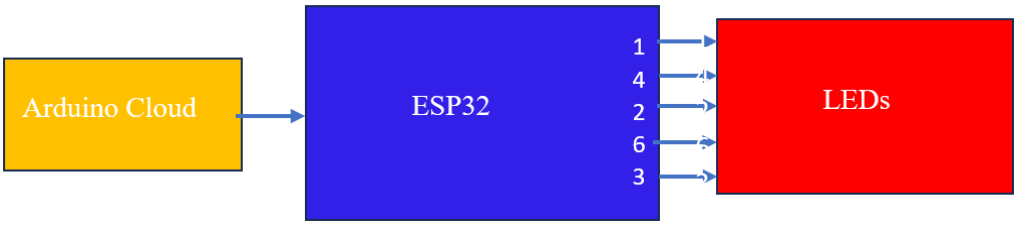


Figure 3.6 Operation Procedure of IoT Control Lighting System

The IoT-based lighting system using ESP32, Arduino Cloud Remote apps, and LEDs operates by integrating the ESP32 microcontroller, LEDs, and the Arduino Cloud platform. Initially, the ESP32 connects to a Wi-Fi network and establishes communication with the Arduino Cloud. LEDs are connected to the ESP32, typically via GPIO pins. Through the Arduino Cloud Remote app, users can remotely control the lighting system. The app sends commands or instructions, such as turning on, off LEDs to the ESP32 via the Arduino Cloud. The ESP32 interprets these commands and accordingly controls the LEDs, enabling the desired lighting effects. This system enables users to remotely manage and customize the lighting settings using the app interface, facilitating convenient and flexible control over the lighting system in real-time from anywhere with an internet connection.

CHAPTER 4
CONCLUSION AND DISCUSSION

The completion of the comprehensive smart home project encompassing safety measures, convenience features, and monitoring systems utilizing Arduino Uno and ESP32 microcontrollers, along with Blynk Cloud and Arduino Cloud, signifies the establishment of a multifaceted and secure home automation ecosystem. Incorporating an automatic fire alarm system ensures heightened safety, alerting inhabitants in case of fire emergencies. The water level control feature offers convenience by managing water usage and preventing overflow. The lighting system not only adds to convenience but also aids in energy conservation through automation. Real-time monitoring of temperature and humidity levels ensures a comfortable living environment while IoT-controlled door locks provide enhanced security. Blynk Cloud facilitates convenient and real-time monitoring of these systems, granting users easy access to vital information and alerts. Meanwhile, Arduino Cloud empowers users with centralized control over door lock for home security and lighting for efficient energy management and ease access. This project amalgamates safety, convenience, and efficiency, laying the groundwork for a smart home that prioritizes security, comfort, and sustainability.

Expanding the value of this project in the real world involves integrating it within smart city initiatives, leveraging its capabilities to enhance urban living. Collaborating with city authorities, this technology can be scaled up to contribute to broader infrastructural improvements such as smart grid integration, optimizing energy distribution, and reducing overall carbon footprint. Implementing traffic management systems, waste management solutions, and air quality monitoring could enhance city-wide efficiency and sustainability. Furthermore, partnerships with businesses or public institutions could utilize this technology to create smart office spaces, educational institutions, or healthcare facilities, fostering innovation, productivity, and safety. Ultimately, embedding this project within the framework of a smart city initiative presents opportunities for widespread societal impact, environmental conservation, and overall improvement in quality of life for urban residents.

APPENDIX A

Components List and Price

No	Name	Price	Quantity	Amount
Central Administrative System				
1	Arduino Uno R3	33,000	1	33,000
2	ESP32	19,000	1	19,000
52,000				
Automation				
Automatic Fire Alarm System				
1	MQ2 Smoke Sensor	4,000	1	4,000
2	Flame Sensor	1,500	1	1,500
3	Buzzer Module	1,000	1	1,000
6,500				
Automatic Water Level System				
1	Ultrasonic Sensor	3,000	1	3,000
2	5V DC Pump Motor	3,000	1	3,000
3	L293D Module	3,500	1	3,500
9,500				
Automatic Lighting System				
1	LDR Module	2,000	1	2,000
2	PIR Motion Sensor	4,000	1	4,000
3	LED	1,700	11	18,700
24,700				
IoT Control				
IoT Based Temperature Monitoring System				
1	DHT 11 Sensor	4,000	1	4,000

4,000				
IoT Administrative Door Lock System				
1	Servo Motor	5,000	1	5,000
5,000				
IoT Control Lighting System				
1	LED	1,700	9	15,300
15,300				
Power Supply				
1	5V Switching Power Supply	5,000	1	5,000
5,000				
Budling Infrastructure				
1	Glitter+screw+Accessories			38500
2	ခဆး			21500
3	PVC+ Hollo + Accessories			120000
Smart Home Project Budget – 302000				

APPENDIX B

Sketch for Uno R3

```
#include "fire_alarm.h"
#include "water_level_controller.h"
#include "lighting_control.h"
#include "humidity_n_temperature_info.h"
#include "iot_export.h"
void setup ()
{
  Serial.begin(9600);
  fire_alarm_setup();
  water_level_controller_setup();
  lighting_control_setup();
  humidity_n_temperature_info_setup();
}
void loop ()
{
  fire_alarm_loop();
  water_level_controller_loop();
  lighting_control_loop();
  humidity_n_temperature_info_loop();
  iot_export_loop();
}
#ifndef FIRE_ALARM_H
#define FIRE_ALARM_H
const int buzzer_pin {2};            // digital
const int flame_sensor_pin {3};      // digital
const int gas_sensor_pin {A0};       // analog
const int gas_threshold {20};
bool fire_alarm_status {false};
void fire_alarm_setup ()
{
  pinMode(buzzer_pin, OUTPUT);
  pinMode(gas_sensor_pin, INPUT);
  pinMode(flame_sensor_pin, INPUT);
```

```
}
void fire_alarm_loop ()
{
 int detected_gas {analogRead(gas_sensor_pin)};
 bool detected_flame {digitalRead(flame_sensor_pin)};      // flame detection > LOW | no flame detection > HIGH...yeh it is in reverse
  // Serial.println("Fire alarm Info : ");
  // Serial.println(detected_gas);
  // Serial.println(detected_flame);
 if ((detected_gas > gas_threshold) && (detected_flame == LOW)) {
  tone(buzzer_pin, 5000);   fire_alarm_status = true;
 }
 else {
  noTone(buzzer_pin);       fire_alarm_status = false;
 }
}
#endif
#ifndef HUMIDITY_N_TEMPERATURE_INFO_H
#define HUMIDITY_N_TEMPERATURE_INFO_H
#include <dht11.h>
const int dhtpin {13};            // digital
int dhtread {0};
float humidity {0.0f};
float temperature {0.0f};
dht11 DHT11;
void humidity_n_temperature_info_setup ()
{
}
void humidity_n_temperature_info_loop ()
{
 dhtread = DHT11.read(dhtpin);
 temperature = DHT11.temperature;
 humidity = DHT11.humidity;
  //Serial.print("Humidity (%): ");
  //Serial.println(humidity, 2);
```

```
  //Serial.print("Temperature  (C): ");
  //Serial.println(temperature, 2);
}
#endif
#ifndef IOT_EXPORT_H
#define IOT_EXPORT_H
#include "fire_alarm.h"
#include "water_level_controller.h"
#include "humidity_n_temperature_info.h"
#include <string.h>
/*
int count{0};
int mapped_value {0};
void pseudo_water_level ()
{
 count++;
 if (count == 40)
   count = 0;
 mapped_value = map(count, 0, 40, 1, 100);
}
*/
void iot_export_loop ()
{
 //pseudo_water_level();
 String data {""};
 data += (int)temperature;
 data += ',';
 data += (int)humidity;
 data += ',';
 data += (100 - ((int)map(water_level, 0, 16, 1, 100)));
 data += ',';
 data += (int)fire_alarm_status;
 data += ',';
 data += "EOS";
 Serial.print(data);
```

```
  delay(1500);
}
#endif
#ifndef LIGHTING_CONTROL_H
#define LIGHTING_CONTROL_H
const int motion_sensor_pin {8};          // digital
const int pir_light_pin {9};              // digital
const int ldr_pin {10};                   // digital
const int ldr_light_pin_batch1 {11};      // digital
const int ldr_light_pin_batch2 {12};      // digital
int timer_count {0};
int reset_counter {0};
bool timer_start {false};
void timer ();
void ldr_output_control ();
void pir_output_control ();
void lighting_control_setup ()
{
  pinMode(pir_light_pin, OUTPUT);
  pinMode(ldr_light_pin_batch1, OUTPUT);
  pinMode(ldr_light_pin_batch2, OUTPUT);
  pinMode(ldr_pin, INPUT);
  pinMode(motion_sensor_pin, INPUT);
}
void lighting_control_loop ()
{
   //Serial.print("Lighting Control Info : ");
   //Serial.print(timer_count);
   //Serial.print(" : ");
   //Serial.println(reset_counter);
  timer_count++;
  pir_output_control();
  ldr_output_control();
}
void ldr_output_control ()
```

```
{
 if (digitalRead(ldr_pin)) {
  digitalWrite(ldr_light_pin_batch1, HIGH);
  digitalWrite(ldr_light_pin_batch2, HIGH);
 }
 else {
  digitalWrite(ldr_light_pin_batch1, LOW);
  digitalWrite(ldr_light_pin_batch2, LOW);
 }
}
void pir_output_control ()
{
 if (timer_count == 40) {                          // restart the timer
  timer_count = 0;
 }
 if (digitalRead(motion_sensor_pin)) {                    // start the timer if motion is detected
  timer_start = true;
  reset_counter = 0;
  digitalWrite(pir_light_pin, HIGH);
 }
 if (((timer_count % 2) == 0) && (timer_count != 0)) {          // check every 5 seconds
  //Serial.println("\t\t CHECK \t\t");
  if (!digitalRead(motion_sensor_pin))
{                                                  // timer will advance until it senses
another motion or until it expires
    reset_counter++; timer_count++;
  }
  else {
    reset_counter = 0;                             // restart the timer upon sensing another
motion
  }
 }
 if (reset_counter >= 2) {                          // if timer expires without sensing addition
motion...lights will go off
  digitalWrite(pir_light_pin, LOW);
```

```
  }
}
#endif
#ifndef WATER_LEVEL_CONTROLLER_H
#define WATER_LEVEL_CONTROLLER_H
const int echo_pin {4};            // digital
const int trigger_pin {5};         // digital
const int motor_pin_firdir {6};    // digital     first direction
const int motor_pin_secdir {7};    // digital     second direction
float detected_duration {0};
float water_level {0};
float base_level {15};
float lower_level {10};
float upper_level {3};
void trigger ()
{
  digitalWrite(trigger_pin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigger_pin, LOW);
}
void water_level_controller_setup ()
{
  pinMode(trigger_pin, OUTPUT);
  pinMode(motor_pin_firdir, OUTPUT);
  pinMode(motor_pin_secdir, OUTPUT);
  pinMode(echo_pin, INPUT);
}
void water_level_controller_loop ()
{
  trigger();                                    // generate ultrasonic wave every 10
microseconds
  detected_duration = pulseIn(echo_pin, HIGH);
  water_level = ((detected_duration * 0.0343) / 2);
  // Serial.print("Water Level Controller Info : ");
  //Serial.println(water_level);
```

```
  if (water_level >= base_level || water_level >= lower_level) {
    digitalWrite(motor_pin_firdir, HIGH);
    digitalWrite(motor_pin_secdir, LOW);
  }
  else if (water_level <= upper_level) {
    digitalWrite(motor_pin_firdir, LOW);
    digitalWrite(motor_pin_secdir, LOW);
  }
}
#endif
```

Sketch for ESP32

```
#define BLYNK_TEMPLATE_ID "TMPL6Uc1gsVfY"
#define BLYNK_TEMPLATE_NAME "InsightNex"
#define BLYNK_AUTH_TOKEN "q8dzw10Jzlrwerg6oBbM0g3_ENGGqyVV"
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include "status_monitor.h"
char ssid[] = "Smaho";
char pass[] = "Pr0jectsmah0";
const int RXp2 {16};
const int TXp2 {17};
#include "arduino_cloud_lights_control.h"
#include "arduino_cloud_door_control.h"
void setup() {
  Serial.begin(115200);
  Serial2.begin(9600, SERIAL_8N1, RXp2, TXp2);
  delay(1500);
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
  initProperties();
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
  arduino_cloud_lights_control_setup();
```

```
  arduino_cloud_door_control_setup();
}
void loop() {
  Blynk.run();
  ArduinoCloud.update();
  lights_control();
  onStatusChange();
  onDoorChange();
  print();
  status_monitor_loop();
}
#ifndef ARDUINO_CLOUD_DOOR_CONTROL_H
#define ARDUINO_CLOUD_DOOR_CONTROL_H
#include <ESP32Servo.h>
#include "thingProperties.h"
Servo door_servo;
int servo_pin {13};
bool door_open {false};
bool door_close {false};
void arduino_cloud_door_control_setup ()
{
  door_servo.attach(servo_pin, 1000, 2000);
}
void onDoorChange ()
{
  if (door && !door_open) {
      door_servo.write(180);
      delay(300);
      door_servo.write(90);
      door_open = true;
      door_close = false;
      Serial.println("DOOR OPEN");
  }
  else if (!door && !door_close) {
    door_servo.write(0);
```

```
    delay(300);
    door_servo.write(90);
    door_open = false;
    door_close = true;
    Serial.println("DOOR CLOSE");
  }
  door_servo.write(90);
}
#endif
#ifndef ARDUINO_CLOUD_LIGHTS_CONTROL_H
#define ARDUINO_CLOUD_LIGHTS_CONTROL_H
#include "thingProperties.h"
int fir {14};
int sec {26};
int thi {33};
int fou {15};
int fif {18};
int six {21};
int sev {27};
int eig {25};
int nin {32};
int ten {19};
int light_array[10] {fir, sec, thi, fou, fif, six, sev, eig, nin, ten};
bool switch_array[10] {LOW, LOW, LOW, LOW, LOW, LOW, LOW, LOW, LOW, LOW};
void onLightIdChange () { };
void onLightSwitchChange () { };
void onMainSwitchChange () { };
void onStatusChange()
{
  status = switch_array[light_id];
  light_switch = switch_array[light_id];
}
void arduino_cloud_lights_control_setup ()
{
  for (int index{0}; index < 10; index++)
```

```
    pinMode(light_array[index], OUTPUT);
}
void lights_control ()
{
  if (light_switch) {
    switch_array[light_id] = HIGH;
    digitalWrite(light_array[light_id], HIGH);
  }
  else {
    switch_array[light_id] = LOW;
    digitalWrite(light_array[light_id], LOW);
  }
}
void print ()
{
  for (int index{0}; index < 10; index++)
  {
    Serial.print(light_array[index]);
    Serial.print(" : ");
    Serial.print(switch_array[index]);
    Serial.print(" | ");
  }
    Serial.println (" ");
}
#endif
#ifndef STATUS_MONITOR_H
#define STATUS_MONITOR_H
#include <string.h>
int data_stream[] {0, 0, 0, 0, 0};
void strtoint_converter (String buffer, int& intval)
{
  //Serial.println("strtoint_converter");
  //Serial.println(buffer);
  intval = 0;
  int index {0};
```

```
  while (buffer[index] != '\0')
  {
   switch (buffer[index])
   {
     case '0': (index != 1) ? intval += 0 : intval += 0;        break;
     case '1': (index != 1) ? intval += 10 : intval += 1;       break;
     case '2': (index != 1) ? intval += 20 : intval += 2;       break;
     case '3': (index != 1) ? intval += 30 : intval += 3;       break;
     case '4': (index != 1) ? intval += 40 : intval += 4;       break;
     case '5': (index != 1) ? intval += 50 : intval += 5;       break;
     case '6': (index != 1) ? intval += 60 : intval += 6;       break;
     case '7': (index != 1) ? intval += 70 : intval += 7;       break;
     case '8': (index != 1) ? intval += 80 : intval += 8;       break;
     case '9': (index != 1) ? intval += 90 : intval += 9;       break;
   }
   index++;
  }
  //Serial.println(intval);
}
void decode_string (String raw_information)
{
  int advance {0};
  int index {0};
  String buffer {""};
  while (raw_information[advance] != '\0')
  {
   if (raw_information[advance] != ',') {
     buffer += raw_information[advance];
   }
   else if (raw_information[advance] == ',') {
     strtoint_converter(buffer, data_stream[index]);
     buffer = "";
     index++;
     //Serial.println(data_stream[index]);
   }
```

```
    //Serial.println(raw_information[advance]);
    advance++;
  }
}
void status_monitor_loop ()
{
  String incoming_information {Serial2.readString()};
  decode_string (incoming_information);
  Serial.println("Status Monitor");
  Serial.println(incoming_information);
  for (int count{0}; count < 5; count++)
    Serial.println(data_stream[count]);
  Blynk.virtualWrite(V0, data_stream[0]);
  Blynk.virtualWrite(V1, data_stream[1]);
  Blynk.virtualWrite(V2, data_stream[2]);
  Blynk.virtualWrite(V3, data_stream[3]);
}
void message_loop ()
{
  Serial.println("MESSAGE RECEIVED: ");
  Serial.println(Serial2.readString());
}
#endif
#ifndef THINGPROPERTIES_H
#define THINGPROPERTIES_H
#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>
const char DEVICE_LOGIN_NAME[]  = "885314d5-5d3d-4d73-9d23-98477e245a31";
const char DEVICE_KEY[]        = "XIZVHEHNMR2CCWHR5QJ9";    // Secret device
password
const char SSID[]              = "Smaho";    // Network SSID (name)
const char PASS[]              = "Pr0jectsmah0";    // Network password (use for WPA, or use as
key for WEP)
void print ();
void lights_control ();
```

```
void onDoorChange ();
void onStatusChange ();
void onLightIdChange ();
void onLightSwitchChange ();
void onMainSwitchChange ();
void onDoorChange();
int light_id;
bool light_switch;
bool status;
bool door;
bool main_switch;
void initProperties(){
 ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(light_id, READWRITE, ON_CHANGE, onLightIdChange, 5);
  ArduinoCloud.addProperty(light_switch, READWRITE, ON_CHANGE,
onLightSwitchChange);
  ArduinoCloud.addProperty(status, READWRITE, ON_CHANGE, onStatusChange);
  ArduinoCloud.addProperty(door, READWRITE, ON_CHANGE, onDoorChange);
  ArduinoCloud.addProperty(main_switch, READWRITE, ON_CHANGE,
onMainSwitchChange);
}
WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
#endif
```

APPENDIX C
PROJECT RECORD